Cognitive levels and Software Maintenance Sub-tasks

Tara Kelly
Limerick Institute of Technology
Tara.Kelly@staffmail.ul.ie

Jim Buckley
Lero / CSIS Department
University of Limerick
Jim.buckley@lero.ie

Keywords: POP-IIB program comprehension, maintenance POP-VB observation

Abstract

This paper describes a case study that was carried out to characterize the behaviour of professional programmers, working on in-vivo software maintenance tasks, in terms of the cognitive levels of Bloom's taxonomy. Specifically, it evaluates if their behaviour on specific maintenance sub-tasks can be associated with specific cognitive levels of the taxonomy.

The findings suggest that some such relationships do exist. Indeed, several of the identified relationships are at the most cognitively-demanding levels of Bloom's taxonomy. Allied with reports from other research in the domain of software maintenance, these relationships suggest that difficult sub-tasks within software maintenance are difficult, in part, because of the cognitive levels that programmers must work at when undertaking them. Thus, they suggest the nature of the support that should be offered to maintenance programmers involved in these tasks.

1. Introduction

Software maintenance is typically the longest phase in a successful software system's lifecycle, as it stretches from the point of initial deployment until the system is taken out of service though various release iterations [1]. Hence, research has suggested that maintenance can consume up to 90% of total systems' costs [2]. In fact, Sommerville [5] states that 75% of software engineers perform maintenance more often than any other work activity.

Reports suggest that half of this effort can be directly related to understanding the software [3], [4]. Because of these figures, many researchers have studied how programmers behave as they interact with the software systems they study. These research areas include information-seeking [6], [7], [8], [9], concept location [10], [11], [12], [13], [14] and software comprehension. Software comprehension can be defined as "the process in which the programmer uses prior knowledge about programming, and information present in the program, to form a dynamic (and static) model of the program which can then be applied to a task" ([15], pg 14). Central to this is an assimilation process which uses programmers' existing knowledge and external representations of the system to augment their mental representation of that system [16], [17], [18], [19], [20], [21].

Closely related to software comprehension is the concept of cognitive levels: the degree to which programmers demonstrate cognitive mastery/skill over the software systems they work on. Recently researchers have begun probing this aspect of software comprehension using Bloom's taxonomy [22], [23], [24], [25], [26]. Bloom's taxonomy is a framework adopted from the educational domain, where assessing cognitive ability and skill is a core domain objective [27]. The taxonomy classifies cognitive skills into a six-tiered hierarchy where, Bloom states, each level builds on the cognitive skills in the preceding levels and thus demonstrates educational advancement.

Several researchers have proposed this framework as relevant for software maintenance [22], [23], [24], [25], some even suggesting it as a 'contextualizing whole' for software comprehension [23]. For example, Xu and Rajlich [22] found all the elements of Bloom’s taxonomy in programmer’s debugging behaviour.

We argue that, as higher-level cognitive skills seem the most demanding, it would be interesting to determine if there are relationships between these high-level cognitive skills and specific maintenance subtasks. If such relationships could be identified, it could suggest the sub-tasks within maintenance that require the most cognitive effort and thus the subtasks that require the most support. In addition, by identifying the cognitive levels, it would also suggest the nature of the support necessary for these subtasks. Finally, such relationships might be used to direct the education of programmers in terms of the cognitive skills relevant for their professional lives.

This, paper reports on a case study that uses a context-aware analysis schema to classify the utterances made by professional programmers (as they work on real-life software maintenance tasks) into the cognitive levels described by Bloom. Using the results of this classification, areas where there was a specific emphasis on a particular cognitive level were studied in depth, to identify the maintenance subtasks being performed at that time. As a result of this analysis, several interesting relationships are identified.

Section 2 in this paper reviews Bloom’s taxonomy, illustrating its traditional role in education and highlighting how this traditional role is adapted to the context of this study. Section 3 describes the case study, detailing the research question, the empirical context, the protocol employed and the data analysis protocol. Section 4 presents the results, showing that all programmers worked at all levels of the taxonomy and that a number of relationships seemed to exist between specific programmers’ maintenance activities and specific cognitive levels of the taxonomy. Finally section 5 discusses these findings.

2. Bloom’s Taxonomy

At a convention for the American Psychological Association in 1948 a group was formed to develop a framework that could be used by all examiners and educators to evaluate student learning in a systematic manner. The result was a taxonomy, called Bloom’s taxonomy, in three verticals – the cognitive, the affective, and the psychomotor domains [28]. The cognitive domain was proposed to assess the cognitive skills or mastery that students display over learnt material and thus is of most relevance to this work.

The cognitive taxonomy itself is placed in a hierarchy which contains six levels, as can be seen in Figure 1. Each new level represents an increase in cognitive skill from the baseline of Knowledge, to the highest level of Evaluation. These 6 levels are briefly reviewed here from a ‘programmer’ perspective but, for a more detailed description, the interested reader is referred to Bloom’s original work [27]:

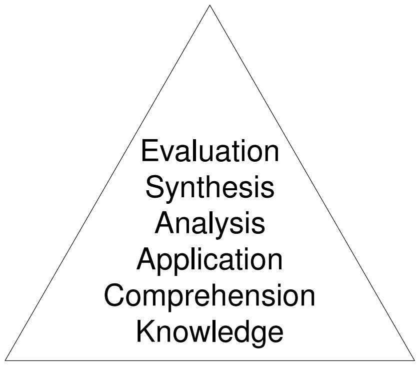

Fig 1: Blooms Taxonomy.

- *Knowledge* deals with the cognitive ability of recall. An example would be when a programmer recalls the correct structure of an 'if' statement.
- *Comprehension* uses knowledge to understand a communication and translate it. A programmer documenting an algorithm shows comprehension.
- *Application* involves making well-based changes to existing communications. An example would be when a programmer debugs or enhances a piece of code.
- *Analysis* concentrates on breaking a communication down into its (often implicit) component parts, and identifying their relationships. An example would be where a programmer recovers a software system's architecture.
- *Synthesis* refers to the ability to build something new from existing Knowledge, Comprehension and Analysis. A relevant example would be the ability to build a new computer application.
- *Evaluation* refers to the ability to make judgements about ideas, work, material and solutions. An example would be the ability to critically review design alternatives for a software system, in the light of, possibly conflicting, requirements.

As suggested by Figure 1, the taxonomy was proposed as a cumulative hierarchy where higher levels were based on the cognitive skills shown at lower levels. This implies that the cognitive skills required at elevated levels are both more complex and less prevalent.

The resultant taxonomy has been translated into 21 languages [29] and has been used world-wide [30], [31], [32], [33]. At one point the taxonomy was the most frequently cited course for educational research [28] and the Social Science Citation Index includes hundreds of articles referencing it [34]. So it can be said that, over the decades, this taxonomy has become the de-facto standard for measuring the cognitive skill which students demonstrate with respect to their learnt material.

In the context of this work, Bloom's taxonomy is being used in a similar, but slightly different fashion. It still characterizes the cognitive skills which professional programmers employ over their "learnt material" (the software systems they are maintaining) during real-world software maintenance tasks. But instead of measuring the *correctness* of programmers' cognitive skills, as would traditionally happen in education, it is used here to assess the *amount of time* that professional programmers work at each specific cognitive level. That is, the amount of utterances these programmers make at each level is assessed, giving a proxy for the amount of effort the programmers expend at each level. Hence, it is used to assess the amount of effort programmers expend at each cognitive level with respect to the maintenance subtasks they perform.

3. The Case Study

3.1 The Research Question

The case study was designed to evaluate if specific cognitive levels are associated with specific maintenance sub-tasks, as performed by professional programmers during real-world maintenance. The protocol used to evaluate this question can be summarized as follows:

- Gather the think-aloud data from a cohort of professional programmers as they perform the maintenance tasks assigned to them by their line managers.
- Segregate the data gathered into discrete utterances and classify each one into the appropriate level of Bloom's taxonomy.

- Analyse the data for clusters where specific cognitive levels are emphasized and review the talk-aloud data, in depth, to classify the maintenance sub-task that is being performed at that time.

More detail on the case study protocol is now provided.

3.2 The Participants and the Systems worked on

The 6 participants employed in the study were all recruited through a series of visits to software companies, and software departments in large organizations. All these professional-programmer participants were self-volunteering males, and ranged in age from 22 to 47 (mean: 35.6 years). They all had at least 3 years industrial programming experience in their current language (mean: 5.6 years) and all except one participant had industrial experience in at least 2 different programming languages (mean: 3 languages). While one programmer was only employed in his current company for 0.5 years, the mean across participants was 6 years.

The systems these programmers worked on ranged from 1 KLOC to 1.5 MLOC (mean 377 KLOC), and were all either state-sector information systems (tasks 1-4) or commercial software applications (tasks 5-8).

3.3 The Maintenance Tasks

Participants 1 and 2 performed 2 tasks each during the study, each assigned to them by their line manager. Participants 3-6 each performed 1 task during the study, again assigned to them by their line manager. These 8 tasks were part of their daily workload. The first 4 tasks came from the health informatics domain. They were to: export data to another system (tasks 1 and 3 addressed different aspects of this goal), heighten the consistency of the current information in the database (task 2) and issue more-targeted renewal forms for disability payments (task 4). Of the other tasks, 3 could be considered functional enhancements to existing applications in the insurance domain (task 5), the travel domain (task 6) and the sales domain (task 8). The last task (task 7) could be considered a configuration enhancement, as it involved adding the ability to select additional printers for an existing application. (Note that participant 1 performed maintenance tasks 1 and 2 and participant 2 performed maintenance tasks 3 and 4. The other participants, 3, 4, 5, and 6, performed tasks 5, 6, 7, and 8 respectively).

Participants	**Domain**	**Task**
1	Healthcare	Export data to another system
	Healthcare	Heighten the consistency of the current information in the database
2	Healthcare	Export data to another system
	Healthcare	Issue more targeted renewal forms for disability payments
3	Insurance	Functional enhancements to existing applications.
4	Travel	Functional enhancements to existing applications;
5	Travel	Adding an additional printer to various desktops
6	Sales	Functional enhancements to existing applications in the travel domain

Table 1: The Maintenance Tasks

3.4 The Case Study Protocol

After initial recruitment, the participants rang us in advance of performing a maintenance task which they had been assigned. On the day they planned to do the assigned task we arranged to visit their workplace. Before the session commenced, participant, task and system information was gathered using a pre-session questionnaire.

Then, for a 2-hour period, for each task, the participants were asked to wear a small MP3 recorder and talk as they worked. All but one of the best-practice guidelines of Ericsson and Simon were followed when gathering this think-aloud data [35]. That is, programmers were asked to say *everything* that came into their mind *as* it came into their mind. Contrary to the best practice guidelines, participants were not prompted when they fell silent, as suggested by Ericsson and Simon. This was because the researcher left the room for each session, in order to heighten the ecological validity of the study. However overall, programmers fell silent only 6.3% of the time, suggesting that this protocol captured the best approximation of their mental state [36] for the vast majority of the time.

After the maintenance session, the programmers were asked if they wished to change any comments they had made on the pre-session questionnaire, in hindsight. None did so. Then the generated data was taken for analysis.

3.5 Data Analysis

After transcription, the data-set was broken into individual phrases, resulting in over 6000 utterances. In the past, two main protocols would have been used to analyse data with respect to Bloom's taxonomy. One is a verb-table approach, as used by [22], [24] where the verbs identified in each utterance are used to classify the utterance onto a cognitive level. However, this approach has been shown to be somewhat reductionist, ambiguous, and has a tendency to leave large amounts of the data set un-coded inappropriately [38].

Consequently Kelly's context aware classification schema was chosen. This is a classification schema which has been shown to be less reductionist in nature and less ambiguous while also resulting in a higher proportion of the data set being coded appropriately [38]. It also has the advantage that it is accompanied by an 'open' coding manual which, in accordance with best practice principles [15], provides guidelines, examples and a decision tree in support of the classification process[1]

This context aware classification schema guided the classification of the data into 5 of the 6 levels of Blooms taxonomy only. Synthesis, a level which reports on the construction of something completely new, was deemed irrelevant in the context of software maintenance. This was because software maintenance refers to the fixing or enhancing of the existing software system. Hence Synthesis was omitted from the coding schema and from the classification process carried out in this study. (However McMeekin et al [26], have since added Synthesis to the coding manual for their study of software development)

The improved classification scheme does come at a cost. Consider, for example, the following partial guidelines for classifying Comprehension and Analysis utterances, as taken from the coding manual:

- **Comprehension** – This is where the programmer is attempting to understand the code. In the think-aloud data this could be represented by a programmer attempting to understand the code in a more general nature than a line-by-line basis. So, this is where a programmer would use their Knowledge of the code to build a more complete picture of what the code does overall. This could be seen in programmers talking about code in general terms rather than referring to particular items present in the code. If the program is being summarised or actions performed by the code are identified in terms of their algorithm then this demonstrates Comprehension.

[1] (available at http://www.csis.ul.ie/staff/jimbuckley/).

- **Analysis** – “Analysis emphasizes the breakdown of the material into its constituent parts and detection of the relationships of the parts and of the way they are organised” [27]. This can be related to where a programmer is attempting to understand the relationships in a software system or program. The programmer may refer to delocalised elements and may specifically discuss them in terms of something encompassing such as the system’s functionality or architecture.

As can be seen from these descriptions, it can be quite easy to confuse categories. For example, ‘Comprehending’ the code, may result in the programmer building ‘Analysis’-type relationships between the code and its domain functionality. So there is a difficulty in categorizing such utterances (in this study we classified them as ‘Analysis’ as they did not represent translation but instead represented relationships between 2 different domains).

Such descriptions illustrate that the analysis schema is work intensive and prone to low reliability. As a result, a series of reliability analyses [37] were performed in advance of the full scale analysis, using 2 independent coders and small samples (4 * 30 utterances) of the data set. Only when the reliability was reasonable (Cohen’s Kappa = 0.585) did the first author proceed to code the whole data-set. Half way through this coding process, the reliability assessment was repeated by the 1st author and 2 new, independent coders, on a sample of the data, in order to assess interpreter drift [40]. Again the reliability was found to be reasonable (Cohen’s Kappa=0.579).

After coding all the utterances, the data set was browsed to identify locations where there was an emphasis in processing at specific cognitive levels. These were recorded and subsequently, the think-aloud data in these locations was analysed to identify the tasks that the participants were performing at that point in time.

4. The Results

Table 1 and Table 2 detail the number and percentage of utterance at each level of the taxonomy for each task. The table illustrates that all developers, in all tasks, worked to some degree, at all 5 assessed levels, suggesting that the entire taxonomy (excluding Synthesis) is relevant for real programmers working on real maintenance tasks.

Task	Knowledge	Comp.	Application	Analysis	Evaluation	Total
1	324	281	84	43	25	757
2	127	200	131	72	12	542
3	343	123	180	140	17	803
4	206	90	87	18	5	406
5	106	104	59	71	52	392
6	148	79	46	38	37	348
7	129	40	111	17	45	302
8	38	50	57	13	13	171
Total	1421	967	755	412	206	3721
Avg.	178	121	94	52	26	465

Table 2: Number of utterances at each cognitive level.

Task	Knowledge	Comp.	Application	Analysis	Evaluation
1	28.5%	24.7%	7.4%	3.8%	2.2%
2	14.4%	22.6%	14.8%	8.1%	1.4%
3	25.7%	9.2%	13.5%	10.5%	1.3%

4	30.1%	13.2%	12.7%	2.6%	0.8%
5	16.8%	16.5%	9.4%	11.3%	8.3%
6	28.3%	15.1%	8.8%	7.3%	7.1%
7	17.4%	5.4%	15%	2.3%	6.1%
8	11.7%	15.4%	17.6%	4%	4%

Table 3: Relative % of utterances at each cognitive level.

4.1 Relationships between maintenance sub-tasks and cognitive levels

The subsequent analysis showed several interesting relationships between the cognitive levels that the programmers operated at and the specific maintenance sub-tasks that they performed:

Evaluation occurs during visual testing of code: This was probably the most prevalent relationship identified across the data set. It was apparent in tasks 1, 5, 6, 7 and 8. For example, towards the end of the data set associated with task 5, the programmer started to test a change that he had made. This caused an increase in the Evaluation utterances where the programmer commented on the quality of the code. Likewise after the programmer had made changes in task 1 he started to test and evaluate the work that had just been carried out, again resulting in an increase in evaluation utterances.

Knowledge seems to be widely associated with visual testing and error fixing. In tasks 2, 4, 5, and 7, when the programmers finished making their changes and started to review the changes, they number of knowledge level utterances increased. Thus, not only did they move up the taxonomy to Evaluation, but they also moved down the taxonomy and operated at the Knowledge level. This occurred when they were testing code visually: in other words, they were looking at the changes they made and evaluating if they were correct. In these cases Knowledge tended to be high as they were studying the code line-by-line. For example, in task 5 the programmer moved through the code on a line-by-line basis in order to find the place where the change should be made and, once he found it, he made the changes. Once made, the programmer then checked them by moving through the code line-by-line and evaluating it.

Likewise, in task 4 towards the end of the session there was a very noticeable increase in the number of Knowledge utterances. When the data was examined it could be seen that the programmer had made all the necessary changes to the code and had started to test it. The code did not run correctly and the programmer indicated in the think-aloud that this is probably due to a bug. The programmer started to move through the code line-by-line in order to identify the bug and fix it. Therefore, at the end of the session, the majority of utterances were at the Knowledge level. However, it should be noted that when the programmers chose to use a testing tool in order to automate testing, this high level of Knowledge was not necessarily present.

Analysis seems linked to code changes. This can be seen in tasks 2, 3, 5 and 8. Often, when Application utterances were elevated so were Analysis level utterances. This would suggest that making changes to code may prompt a programmer to move to a higher level of cognition, possibly to analyse how the change will impact on delocalised elements of the system. For example, in task 2 when there was an increase in the Analysis type utterances, the data showed that the programmer was making changes to the code and was relating these changes back to other parts of the domain and the system. He was thereby recognising the relationship between this code and other elements. Likewise in task 8, there was a fairly consistent number of Analysis level utterances, during the first half of the session when changes were being made. The code that had been changed, had connections to databases, and the programmer often referred to this and the impact the code changes had on them.

Finally, it is also interesting to note that amount of Analysis level processing was relatively high in Task 5 and (as mentioned above) the first part of task 8. Task 5 involved the creation of a new testregion, where new and altered functionality could be tested against a complete data-set before going 'live'. This task, by its very nature is delocalized, as the test area must accommodate all of the existing system. Likewise, task 5 involved the amendment of code that connected to databases, again a task delocalized in nature. This suggests that it is not just the maintenance subtasks that impact on the level at which the participants processed at but also, in some instances, that the encompassing maintenance task can be characterized by cognitive level.

5. Results Discussion

From these observations it seems that programmers don't just move randomly through the levels of cognition during a maintenance task. What is being undertaken at any point in time seems to have an impact on the level of cognition being used.

What is particular interesting about the trends identified is that 3 of the five trends are at elevated levels of the taxonomy, levels that Bloom argued were the most complex. Evaluation was shown to be associated with visual testing of code, after changes were made. Likewise Analysis was shown to be associated with assessing the impact of code changes.

This mirrors some of the work done in the information seeking domain. For example Ko, in his characterization of co-located commercial programmers [9], suggests that some of these programmers most prevalent, yet unavailable, information needs are to identify the impact of a change (d4) and to identify the strategically related code (u2). This relates directly to the increase in Analysis level utterances found in this data set, during code changes. It also comments on the difficulty of obtaining these Analysis level insights.

Likewise Sharif and Buckley [41], in their characterization of Open-Source programmer's information needs, suggest that programmers frequently seek validation from others to evaluate their work after they have completed maintenance changes. Again this attests to the prevalence of the association and the difficulty of the task.

In terms of the implications of such findings, the difficulty of working at elevated cognitive levels that Bloom reports and the relationships between these levels and specific maintenance subtasks suggests a characterization of maintenance sub-tasks in terms of the relative effort they consume. Checking code quality and assessing the impact of changes would seem like high-effort activities. Likewise maintenance tasks that are delocalized in nature, would also seem effort intensive. If further work buttresses these findings, then it provides a focus for visualization tool providers as to the maintenance tasks they should be supporting. It suggests they should be providing navigation mechanisms around delocalized elements of the system and that there should be tool support for evaluation of code segments.

It also provides a focus for educators as to the type of cognitive skills they should train maintenance programmers in. To enable theses programmers' efficiency at these tasks, educators should focus on evaluation skills and analysis level skills in their curricula.

Unfortunately though, the results presented here should be viewed with some trepidation, given the small sample size, the self-selecting population and the 'reasonable' level of reliability obtained with the coding manual. This notwithstanding, it is the most comprehensive data set to date in the study of cognitive level usage by professional programmers involved in real-world software maintenance. In fact, it is the only such study that employs more than 1 professional programmer. In addition, it improves upon other studies in the area through the use of in-vivo data in a range of maintenance contexts, and through the use of the context-aware analysis schema as a measurement tool.

One problem that should be addressed in future work is that there was no measure of 'correctness' for the utterances that the programmers made. If a programmer was commenting on the efficiency of the code, for example, then his utterances were coded as Evaluation, regardless of whether the comments were correct or incorrect. This is important in the context of the current study because, if all the assertions they make at elevated cognitive levels are correct, it suggests that information seeking at this level is trivial. This in turn suggests there is little need for visualization tools to support these tasks or for educators to help train the programmer population.

On the other hand, if many of the utterances they make are incorrect, it does suggest that support needs to be provided. Unfortunately, given the scale of the software systems being maintained and our lack of familiarity with these systems, it would have been impossible for our analysis to determine the correctness of each utterance. However, this element of correctness would seem to report fairly directly on the difficulty of mastering each of the cognitive levels present in the think-aloud data and should be studied by later studies.

7. Conclusions

By classifying the think-aloud data produced by professional programmers during in-vivo software maintenance tasks into the cognitive levels suggested by Bloom, and by performing cluster analysis on these levels, this case study suggested specific relationships between maintenance sub-tasks and specific cognitive levels. It suggested that evaluation was associated with visual inspection of code, that Analysis was associated with code changes and that Knowledge was associated with code changes and error fixing. In addition, it illustrated that some maintenance tasks can be characterized as 'Analysis' oriented in nature.

These findings should prompt further studies to buttress the findings presented here and to extend them. Specifically, additional maintenance subtasks should be addressed, such as finding the location of relevant code [42], or finding the causes of a certain program state or failure [9]. Likewise studies which capture the correctness of programmers' utterances at each level would also be a valuable addition. Such studies would have the ability to report on the cognitive skills that are employed by programmers and the cognitive skills they have difficulties with. Thus they identify the cognitive skills that need to be supported, in terms of education and in terms of visualization.

Acknowledgement

This work was supported, in part, by Science Foundation Ireland grant 03/CE/I303_1 to Lero – The Irish Software Engineering Research Centre (www.lero.ie)

References

[1] Pressman, R.S., (2000), "Software Engineering: A Practitioner's Approach", McGraw-Hill Publishing.

[2] Erlikh, L., (2000), Leveraging legacy system dollars for E-business, (IEEE) IT Pro, May/June, pg 17 – 23.

[3] De Lucia, A., Fasolino, A.R., Munro, M., (1996), Understanding Function Behaviours Through Program Slicing, Proceedings of the 4th Workshop on Program Comprehension, IEEE Computer Society Press, Los Alamitos CA, pg 9-16.

[4] Rajlich, V., (1994), Program Reading and Comprehension. Proceedings of Summer School on Engineering of Existing Software, Dipartimento do Informatica, University of Bari, Italy, pg 161-178

[5] Sommerville, I., (2004), "Software Engineering (7th edition)" Addison-Wesley.

[6] Singer, J., Lethbridge, T., Vinson, N., Anquetil, N., (1997), An examination of software engineering work practices, Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative Research, Toronto, Canada.

[7] Singer, J., (1998), Work practices of software maintenance engineers, Proceedings of the International Conference on Software Maintenance, Washington, Federal District of Columbia, USA, pg 139 – 145

[8] Seaman, C., (2002), The information gathering strategies of software maintainers, International Conference on Software Maintenance.

[9] Ko, A.J., DeLine, R., Venolia, G., (2007), Information Needs in Collocated Software Development Teams, International conference on software engineering (ICSE), Minneapolis, USA.

[10] Rajlich, V., and Wilde, N., (2002), The Role of Concepts in Program Comprehension, Proceedings of the 10th IWPC, Paris, France, IEEE Computer Society.

[11] Rajlich, V., (2001), Role of concepts in software evolution, Proceedings of the International Workshop on Principles of Software Evolution, Vienna, Austria, ACM Press.

[12] Marcus, A., Sergeyev, A., Rajlich, V., Maletic, J.L., (2004), An information retrieval approach to concept location in source code, Proceedings of the 11th Working Conference on Reverse Engineering

[13] Cleary, B., and Exton, C., (2007), Assisting Concept Location in Software Comprehension, 19th Annual Workshop of the Psychology of Programming Interest Group, 2nd – 6th July, University of Joensuu, Finland.

[14] Wilde, N. and Scully, M.C., (1995), Software Reconnaissance: Mapping program features to code, Journal of Software Maintenance: Research and Practice, Vol. 1, No. 7, pg 49 – 62

[15] Good, J., (1999), Programming Paradigms, Information Types and Graphical Representations: Empirical Investigations of Novice Program Comprehension, PhD Thesis, University of Edinburgh, pg. 14.

[16] Brooks, R., (1983), Towards a Theory of the Comprehension of Computer Programs, International Journal of Man-Machine Studies, 18, pg 543-554.

[17] Soloway, E. Bonar, J. Ehrlich, K. (1983), Cognitive Strategies and Looping Constructs: An Empirical Study, Communications of the ACM, November, 26(11), pg 853-860

[18] Letovsky, S., (1986), Cognitive Processes in Program Comprehension, Empirical Studies of Programmers: Proceedings of the 1st Workshop, Ablex:Norwood, pg 58-79.

[19] Von Mayrhauser, A. Vans, A.M. and Lang, S., (1998), Program Comprehension and Enhancement of Software. Proceedings of IFIP World Computing Congress – Information Technology and Knowledge Engineering. August/September, Vienna/Budapest.

[20] Detienne, F., (2002), Software Design: Cognitive Aspects, Springer-Verlang London

[21] Burkhardt, J.M., Detienne, F., Weidenbeck, S., (2002), Object-oriented Program Comprehension: Effect of Expertise, Task and Phase, Empirical Software Engineering, Vol. 7, June, pg 115-156.

[22] Xu, S., Rajlich, V., (2004), Cognitive Processes During Debugging, Proceeding of 3rd IEEE International Conference on Cognitive Informatics, British Columbia, Canada, August.

[23] Buckley, J., Exton, C., (2003), Bloom's Taxonomy: A Framework for Assessing Programmers Knowledge of Software Systems, Proceeding of 11th IEEE International Workshop on Program Comprehension, Portland, USA, May.

[24] Xu, S., Rajlich, V., Marcus, A., (2005), An Empirical Study of Programmer Learning during Incremental Software Development, Proceedings of 4th IEEE International Conference on Cognitive Informatics, California, USA, August.

[25] Lahtinen, E., (2007), A Categorisation of Novice Programmers: A Cluster Analysis Study, Proceedings of the 20th Workshop of the Psychology of Programming Interest Group.

[26] McMeekin, D.A., von Konsky, B.R., Chang, E. and Cooper, D.J.A. (2008), Checklist Inspections and Modifications: Applying Bloom's Taxonomy to Categorise Developer Comprehension, Proceedings of 16th IEEE International Workshop on Program Comprehension, Amsterdam, The Netherlands, June.

[27] Bloom, B.S., Englehart, M.D., Furst, E.J., Hill, W.H., and Krathwohl, D.R., (1956), Taxonomy of educational objectives, Handbook 1: The Cognitive Domain, New York: Longmans, Green.

[28] Bloom, B.S., (1996), Reflections on the Development and Use of the Taxonomy. Bloom's Taxonomy: A Forty-year retrospective, Ed, Anderson, L.W., Sosniak, L.A., History of education quarterly, Vol. 36, No. 1, Spring, pg 1-8.

[29] Krathwohl, D.R., (1996), Reflections on the Taxonomy: Past, Present and Future, Bloom's Taxonomy: A Forty-year retrospective, Ed, Anderson, L.W., Sosniak, L.A., History of education quarterly, Vol. 36, No. 1, Spring, pg 181-202.

[30] Airasian, P.W., (1996), The Impact of the Taxonomy on Testing and Evaluation, Bloom's Taxonomy: A Forty-year retrospective, Ed, Anderson, L.W., Sosniak, L.A., History of education quarterly, Vol. 36, No. 1, Spring, pg 82-102.

[31] Sosniak, L.A. (1996), The Taxonomy, Curriculum and their Relations, Bloom's Taxonomy: A Forty-year retrospective, Ed, Anderson, L.W., Sosniak, L.A., History of education quarterly, Vol. 36, No. 1, Spring, pg 103-125.

[32] Anderson, L.W., (1996), Research on Teaching and Teacher Education, Bloom's Taxonomy: A Forty-year retrospective, Ed, Anderson, L.W., Sosniak, L.A., History of education quarterly, Vol. 36, No. 1, Spring, pg 126-145.

[33] Lewy, A. and Bathory, Z., (1996), The Taxonomy of Education Objectives in Continental Europe, the Mediterranean and the Middle East, Bloom's Taxonomy: A Forty-year retrospective, Ed, Anderson, L.W., Sosniak, L.A., History of education quarterly, Vol. 36, No. 1, Spring, pg 146-163.

[34] Furst, E.J., (1981), Bloom's Taxonomy of Educational Objectives for the Cognitive Domain: Philosophical and Educational Issues, Review of Educational Research, Vol. 51, No. 4, Winter, pg 441-453.

[35] Ericsson, K., Simon, H., (1980), Verbal Reports as Data, Psychological Review, Vol. 89, No. 3, pg 215-251

[36] Russo, J., Johnson, E., Stephens, D., (1989), The Validity of Verbal Protocols, Memory and Cognition, Vol. 17.

[37] Greene, J., and D'Oliveira, M., (1999). Learning to use Statistical Tests in Psychology (2nd ed.). Buckingham, Philadelphia: Open University Press.

[38] Kelly T., and Buckley J..(2006) A Context-Aware Analysis Scheme for Bloom's Taxonomy. Proceedings of the 14th International Conference on Program Comprehension. pp 275-284

[39] Neuendorf, K., (2002), Content Analysis Guidebook, SAGE Publication

[40] Eysenck M.W. and Keane M.T. (19996) Cognitive Psychology: A Student's Handbook 3rd Ed.

[41] Sharif K., and Buckley J.. (2008) Observing Open Source Programmers Information Seeking. Proceedings of the 20th Workshop of the Psychology of Programmers Interest Group. pp 7-16.

[42] Koenemann, J. Robertson, S.P. (1991), Expert Problem Solving Strategies for Program Comprehension, ACM, March, pg 125-130 1(1), 1-3.